Designing Compliant Business Processes with Obligations and Permissions

Stijn Goedertier and Jan Vanthienen

Department of Decision Sciences & Information Management,
Katholieke Universiteit Leuven, Belgium
myFirstName.myLastName@econ.kuleuven.be

Abstract. The sequence and timing constraints on the activities in business processes are an important aspect of business process compliance. To date, these constraints are most often implicitly transcribed into control-flow-based process models. This implicit representation of constraints, however, complicates the verification, validation and reuse in business process design. In this paper, we investigate the use of temporal deontic assignments on activities as a means to declaratively capture the control-flow semantics that reside in business regulations and business policies. In particular, we introduce PENELOPE, a language to express temporal rules about the obligations and permissions in a business interaction, and an algorithm to generate compliant sequence-flow-based process models that can be used in business process design.

1 Motivation and Methodology

Nowadays there is an increased pressure on companies to guarantee compliance of their business processes with business policy, the whole of internally defined business constraints, and business regulations, the whole of externally imposed business constraints. The obligation to guarantee compliance, whether imposed by management, customers, governments or financial markets, is often the main driver for business process automation. The downside to automating business processes, however, is that ill-conceived automation can make business processes more difficult to adapt to ever changing business policies and regulations. As such, automated business processes risk to become in time an impediment to compliance, rather than a enabler. Consequently, reconciling compliance and flexibility is a major concern in business process design.

Companies often only implicitly think about business policy and regulations when they design business processes and pay little attention to avoid hard-coding policies and regulations directly in control-flow based process models. What is lacking is a more declarative approach in business process design in which business policy and regulations are made explicit in terms of definitions and constraints. The sequence and timing constraints on the activities in business processes, known as control flow, are an important aspect of compliance. In a software-release process, for instance, a new version may only be put in production after it has been tested and approved. Similarly, in an order-to-cash

J. Eder, S. Dustdar et al. (Eds.): BPM 2006 Workshops, LNCS 4103, pp. 5–14, 2006.

process, an order may only be shipped by the dispatching office after it has been accepted by a salesperson. Designers often think implicitly about these kinds of permissions and obligations when modeling the control-flow perspective of business processes.

In this paper we show how the logic behind the obligations and permissions can be made explicit in the form of temporal deontic assignments that can be (re)used in business process design. To verify and validate such a set of deontic assignments, we show how to generate a compliant control-flow-based process model from it. The generated process model is not intended for process execution, but can rather be used by the process designer for verification and validation. Moreover, the generated process model allows the designer to identify the decision points and all possible violations of obligations, i.e. exceptions, that can occur.

The remainder of this article is structured as follows. In section 2 we discus the relevant literature on the use of constraints in obtaining business process compliance and flexibility. In section 3, we formally introduce PENELOPE (**P**rocess **EN**tailment from the **EL**icitation of **O**bligations and **PE**rmissions), a language to express temporal deontic assignments. Next, we discuss some issues in the verification and validation of temporal deontic assignments. Finally, in section 5 we define and illustrate the algorithm to generate control-flow based process models from a rule set of obligations and permissions.

2 Related Work

Recently, there is an increased pressure by governments and financial markets on companies to guarantee compliance with corporate governance regulations. Frameworks such as ITIL and COBIT lay down control objectives for specific processes in an IT organization. The Sarbanes-Oxley Act imposes the design of internal controls to prevent fraud for the whole company in general and for the IT organization in particular. Our work focusses on the use of business rules in designing compliant business processes. Different categories of business rules can be used to declaratively specify the control-flow (sequence and timing of activities), data (data validation and requirements) and resource perspectives (task allocation and data access rights) on business processes [1]. In this paper we focus on obtaining a compliant ordering and timing of activities. The subject of compliance, however, is broader than these control flow concerns and comprises issues like the introduction of control steps, separation of duties, the four-eyes principle...

Business process languages such as UML Activity Diagrams, BPMN, Event-Process-Chains, etc. are most often based on the control-flow paradigm, and define an explicit order relation between the activities in the process. These order relations even occur in the case handling paradigm, in which a *preferred* or *normal* control-flow is defined between activities [2]. What is lacking is a declarative approach that makes the partial order relations due to legal requirements more explicit. Bons et al. [3] identify this need to incorporate the legal state into the model of a trade procedure. To this end, the authors propose to annotate

the states in Petri nets with a description of the deontic state. Regulations can be specified between the business partners in a business collaboration (between external agents). In this context regulations are called business protocols [4] or business contracts. Several authors describe a language for intelligent agents to reason about contract state [5] [6] [7] [8]. The objective of this paper differs in that it does not consider the execution-time monitoring of business contracts, but rather considers the impact of sequence and timing constraints on business process design. The issue of compliance is of course also relevant during the diagnosis phase of the BPM life-cycle, in which a conformance check between the sequence and timing constraints and the logged process instances could be possible [9].

3 The PENELOPE Language

Deontic logic is a logic for representing and reasoning about deontic concepts such as obligation, permission, prohibition and waived obligation. Various axiomatizations of deontic logic have been proposed with considerable extensions to Føllesdal and Hilpinen's Standard Deontic logic (SDL) [10]. Broersen et al. use CTL [11] to express the notion of deadline obligations [12]. Several authors have built a Deontic Logic [5] [6] [7] using the Event Calculus formalism, for which Shanahan provides suitable axiomatizations [13]. In these works deontic properties are represented as fluents, such that it is possible to represent and reason about the effects of activities on the obligations and permissions of actors. Table 1 enumerates some of the deontic fluents and axioms we use to represent temporal deontic assignments.

PENELOPE is different from existing languages mainly because it is designed with a purpose to generate compliant control-flow-based process models from a rule set of permissions and obligations. In order to distinguish necessity from possibility in business policy and regulations, the language considers the deontic modalities of obligation, conditional commitment and permission, whereas other languages only consider commitments and conditional commitments [6] [7]. PENELOPE does not consider prohibition or waived obligation. Prohibition is assumed, however, if neither permission nor obligation can be derived. The exclusion of prohibition and waiver prevents a lot of anomalies [14]. Some implementations of Deontic logic interpret deontic assignments as the obligation to bring about a certain proposition, others see it as the obligation to perform a certain activity. Because PENELOPE aims at entailing process models from deontic assignments, activities rather than propositions are the object of deontic assignments. This also allows us to model compound activities such as $Xor(AcceptOrder, RejectOrder)$. Unlike other languages, PENELOPE allows to explicitly define deadlines on the performance of activities in terms of the performance of previous activities. When an agent performs an obligation or permission within due time, the permission or obligation ceases to exist. This is expressed in axioms 1 and 2. Conversely, not performing an obligation within due date leads to a violation, as described in axiom 3. Business policies or regulations might provide so-called reparation, or contrary-to-duty, obligations to deal

with violations [8]. Because we want to capture the semantics of both external regulations and internal business policy, we need to express deontic assignments to both external and internal agents involved in a business interaction. Internal agents in business policies are subordinate to the external agent they represent. Activities performed by an internal agent resort in the same deontic fluents as if they were performed by the representative external agent.

Table 1. Some deontic properties of PENELOPE [14]

term	**meaning**
$Xor(\alpha_1, \alpha_2)$	compound activity $alpha_1$ XOR $alpha_2$
$Or(\alpha_1, \alpha_2)$	compound activity $alpha_1$ OR $alpha_2$
$And(\alpha_1, \alpha_2)$	compound activity $alpha_1$ AND $alpha_2$
$Oblig(\pi, \alpha, \delta)$	partner π must do activity α by due date δ
$Perm(\pi, \alpha, \delta)$	partner π can do activity α prior to due date δ
$CC(\pi, \alpha_1, \delta_1, \alpha_2, \delta_2)$	partner π must do activity α_2 by due date δ_2 after activity α_1 is performed prior to due date δ_1
(1) $Terminates(\alpha, Oblig(\pi, \alpha, \delta), \tau) \leftarrow \tau \leq \delta$	
(2) $Terminates(\alpha, Perm(\pi, \alpha, \delta), \tau) \leftarrow \tau \leq \delta$	
(3) $Happens(Violation(Oblig(\pi, \alpha, \delta)), \delta) \leftarrow HoldsAt((Oblig(\pi, \alpha, \delta)), \delta) \wedge \backsim Happens(\alpha, \delta)$	
(4) $Initiates(\alpha_1, Oblig(\pi, \alpha_2, \delta_2), \tau) \leftarrow \tau \leq \delta_1 \wedge HoldsAt(CC(\pi, \alpha_1, \delta_1, \alpha_2, \delta_2)), \tau)$	
(5) $Terminates(\alpha_1, CC(\pi, \alpha_1, \delta_1, \alpha_2, \delta_2), \tau) \leftarrow \tau \leq \delta_1$	

We give an example to demonstrate the intuition behind PENELOPE. In an order-to-cash business process the external roles of `Buyer` and `Seller` may be distinguished. A seller can have, among others, the internal roles of `Sales`, and `Dispatch`. In addition, the following externally visible activity types could exist: `PlaceOrder`, `AcceptOrder`, `RejectOrder`, `Pay` and `Ship`. For these roles and activities a number of temporal deontic assignments are displayed in Table 2. Assignments 1 to 4 categorize the external business regulation payment-after-shipment, specifying that payment takes place after shipment. Assignment 5, however, is an internal business policy, specifying that no order may be shipped without previously being accepted. These permissions and obligations impose partial order constraints on the activities in a business processes. This set of deontic assignments leads to the process model for both seller and buyer that is displayed in Fig. 1.

4 Verifying and Validating Temporal Constraints

Temporal deontic assignments lay down the rules of interaction either between business partners or among the internal agents of a business partner in particular. Because temporal deontic assignments are the starting point for the design of

Table 2. Payment-after-shipment

natural and formal expression
(1) Initially the buyer has the permission to place an order.
$Initially_p(Perm(Buyer, PlaceOrder(Buyer, Seller)))$
(2) When the seller accepts the order, the seller is committed to pay the seller, one time unit after the seller ships.
$Initiates(AcceptOrder(Seller, Buyer), CC(Buyer, Ship(Seller, Buyer), \delta_s, Pay(Buyer, Seller), \delta_s + 1), \tau)$
(3) When the buyer places an order, the seller must either accept or reject it within one time period
$Initiates(PlaceOrder(Buyer, Seller), Oblig(Seller, Xor(Accept(Seller, Buyer), Reject(Seller, Buyer)), \tau + 1), \tau)$
(4) When the seller accepts the order, the seller must ship within two time units.
$Initiates(Accept(Buyer, Seller), Oblig(Seller, Ship(Seller, Buyer), \tau + 2), \tau)$
(5) Only when sales accepts the order, dispatch may ship the order
$Initiates(Accept(Sales, Buyer), Perm(Dispatch, Ship(Dispatch, Buyer), \delta), \tau)$

a business' private business processes they must be verified and validated. The rich semantics and the availability of efficient reasoning procedures present new opportunities for verification and validation. Without going into detail, we can highlight deadlock, livelock, deontic conflict, temporal conflict and trust conflict verification issues.

In a business interaction each legal scenario must lead to **termination**, a state in which no obligations or permissions exist. In a **deadlocks** situation, no permissible performance can carry the business interaction forward such that a new state of permissions, obligations and conditional commitments exist. Such a scenario might consist of two business partners having conditional commitments towards each other, but the conditional performance to turn at least one of these conditions into a base-level obligation is not permitted. For example, the buyer has made the conditional commitment to pay upon delivery, whereas the seller has made the conditional commitment to deliver upon payment. In a **livelock** situation, the protocol state is trapped in an infinite loop. Notice that it is not the occurrence of a loop that defines the livelock, but the occurrence of loops without a permissible performance that leads to a deontic state outside the loop.

Deontic conflicts arise when there are protocol states in which a business partner has both the permission and the prohibition to a performance or when he has both an obligation and obligation waiver to a performance. Note, however, that it is not possible to have deontic conflicts in PENELOPE, because it does not make use of prohibition and waiver modalities. **Temporal conflicts** occur when two deontic assignments at the same time initiate and terminate a permission, obligation or conditional commitment. In a business interaction **trust conflicts** can also occur. This happens when a business interaction puts the business in a position were it has direct obligations towards non-trusted business partners that involve sensitive activities such as payment or the shipment of goods, that are not neutralized by preceding performances of the opposite party.

5 Generating State Space and Control Flow

In this section we introduce an algorithm do generate the state space of a set of temporal deontic assignments. This state space can be used for verification of the above mentioned anomalies. In addition, this state space can be mapped to control-flow-based process models for each of the business partners in the interaction. Generated process models are not intended for process execution, but can rather be used by the process designer for validation and allow to identify the decision points and possible violations of obligations that can occur.

To generate the process model for a role in a business interaction, one must analyze the temporal obligations and permissions that hold at certain points in time, given certain narratives of activity performances N. To this end, the expressions below define sets of obligations $O(\tau)$ and permissions $P(\tau)$ that hold at state τ, sets of obligations $Od(\delta)$ and permissions $Pd(\delta)$ that are due at state δ and a set of violations without reparation $VWR(\tau)$ that happen at state τ. Notice that a narrative of activity performances N is implicitly assumed in each of these expressions.

$$O(\tau) = \{\alpha : HoldsAt(Oblig(\pi, \alpha, \delta), \tau)\} \tag{1}$$

$$P(\tau) = \{\alpha : HoldsAt(Perm(\pi, \alpha, \delta), \tau)\} \tag{2}$$

$$Od(\delta) = \{\alpha : HoldsAt(Oblig(\pi, \alpha, \delta), \delta)\} \tag{3}$$

$$Pd(\delta) = \{\alpha : HoldsAt(Perm(\pi, \alpha, \delta), \delta)\} \tag{4}$$

$$\begin{aligned} VWR(\tau) = \{\alpha : &Happens(violation(Oblig(\pi, \alpha, \delta)), \tau), \\ &\neg\exists Initiates(violation(Oblig(\pi, \alpha, \delta)), o, \tau)\} \end{aligned} \tag{5}$$

A state τ in our state space corresponds to a set of obligations $O(\tau)$, permissions $P(\tau)$ and conditional commitments $CC(\tau)$ that hold this state. State transitions are defined differently in PENELOPE than in the commitment space defined by Yolum [15]. In PENELOPE a business interaction can move from a state τ_1 to a state τ_2 if there exists a narrative N of permissible performances, between states τ_1 and τ_2, such that the performance of the activities makes the same deontic fluents hold at state τ_2 that are contained by state τ_2. Under the assumption that no cycles can occur in the interaction, the state space can be represented as a directed acyclic graph. To efficiently enumerate a state space beyond a state τ, it suffices to perform all different combinations of permissible performances that are due at the earliest due date of the obligations and permissions that hold in state τ. Figure 2 enumerates the state space of the deontic assignments of the order-to-cash business process in Table 2. A state τ is an end state if no obligations or permissions hold at τ or if there exist violations without reparation.

$$endState(\tau) \Leftrightarrow O(\tau) \cup P(\tau) = \emptyset \vee VWR(\tau) \neq \emptyset \tag{6}$$

Temporal deontic assignments to internal and external agents in a business interaction impose *partial* order constraints on the activities that are carried out. The problem of generation the control-flow for a particular role in a set of

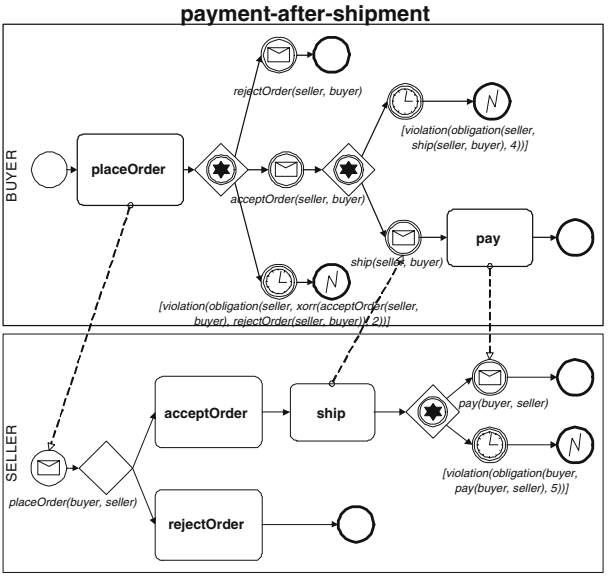

Fig. 1. Two process models generated by PENELOPE

temporal deontic assignments can either be *under specified*, *even specified* or *over specified*. A problem is under specified if no unique sequence flow can be entailed. For even and over specified problems, a unique sequence flow can be derived, provided that the deontic assignments contain no anomalies such as livelocks, deadlocks or contradictions. Given rules 1 to 4 in the example of Table 2, the generation problem is even specified. Adding rule 5 makes the problem over specified, but introduces no contradictions.

We have implemented the PENELOPE language in CLP(fd). In addition we have constructed an algorithm in Prolog to generate a proprietary XML file with the BPMN process model for all external roles in a set of deontic assignments. From this XML file a Microsoft Visio Add-in was written to draw the generated model. We have chosen the BPMN because its visualizations allow us to model external events and exceptions in control-flow. We make use of well-understood and general control flow constructs such as sequence, XOR-split, AND-split, etc. However, due to lack of space and because the process model is intended to facilitate validation for the process designer, we do not clearly formulate the process modeling language used.

The algorithm, of which a summary in pseudo-code is provided below, progressively enumerates all states in the state space and draws the BPMN model for role π. Whenever during state transitions the role π performs activities, this is modeled as a task. Whenever another role performs an activity of which π is a recipient, this is modeled as a message event. The drawing logic of the algorithm is represented by a large number of IF-THEN rules. In the algorithm the obligations and permissions of role π that are due at state δ are contained by

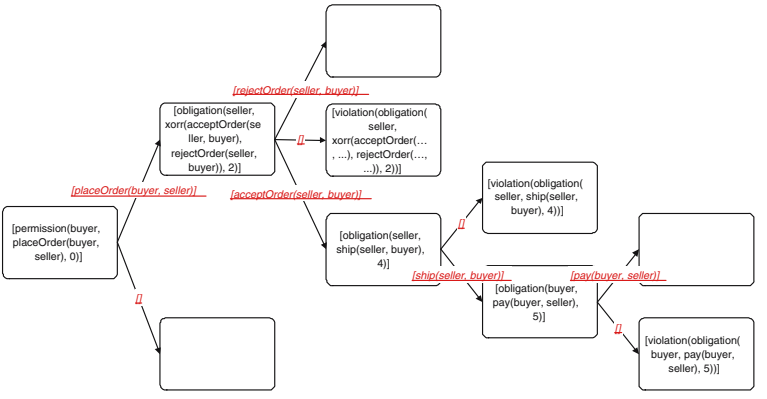

Fig. 2. The state space of the example deontic assignments

the sets $Od(\pi, \delta)$ and $Pd(\pi, \delta)$. A generated process model for a particular role π must not violate the obligations for which no violation is allowed. Therefore, whenever the set $PO(\pi, \delta)$ contains obligations to fulfil, these are drawn as tasks in BPMN. By way of precaution, a generated process model for a particular role π must foresee the possibility that other business partners violate obligations. For instance, when a buyer places an order, he must foresee never to receive a rejection or acceptance from the seller. Violations of obligations can only be detected if the due dates on obligations are timed during process enactment. This is represented in the BPMN model using intermediate timeout events. The obligations and permissions towards role π due at time δ are contained in the sets $OTP(\pi, \delta)$ and $PTM(\pi, \delta)$. In a state in which a violation occurs for which no reparation exists, an error end event is drawn. Notice that process design can only identify exceptions, it is up to the process modeler to properly deal with them. In some cases, the deontic conflicts between business partners might be resolved through human interaction.

1 $PO(\pi, \delta) = \{\alpha : HoldsAt(Oblig(\pi, \alpha, \delta), \delta)\}$
2 $PP(\pi, \delta) = \{\alpha : HoldsAt(Perm(\pi, \alpha, \delta), \delta)\}$
3 $OTP(\pi, \delta) = \{\alpha : HoldsAt(Oblig(\phi, \alpha, \delta), \delta), recipient(\alpha) = \pi\}$
4 $PTP(\pi, \delta) = \{\alpha : HoldsAt(Perm(\phi, \alpha, \delta), \delta), recipient(\alpha) = \pi\}$
5 $OO(\pi, \delta) = \{\alpha : HoldsAt(Oblig(\phi, \alpha, \delta), \delta), \phi \neq \pi\}$
6 $OP(\pi, \delta) = \{\alpha : HoldsAt(Perm(\phi, \alpha, \delta),\ \delta), \phi \neq \pi\}$
7 **drawControlFlow**(π, τ)
8 **if** $\neg endState(S(\tau))$ **then**
9 $\quad \delta \leftarrow earliestDueDate(\tau)$
10 $\quad$ **if** $\{\alpha : \alpha \in PO(\pi, \delta), atomic(\alpha)\} \neq \emptyset$ **then** draw tasks in sequence
11 $\quad$ **if** $\{and(\alpha_1, \alpha_2) : and(\alpha_1, \alpha_2) \in PO(\pi, \delta)\} \neq \emptyset$ **then** draw tasks in parallel
12 $\quad$ **if** $\exists xor(\alpha_1, \alpha_2) \in PO(\pi, \delta)$ **or** $PP(\pi, \delta) \neq \emptyset$ **then** draw XOR gateway
13 $\quad ACs \leftarrow allCombinations(OO(\pi, \delta) \cup OP(\pi, \delta) \cup PP(\pi, \delta))$
14 $\quad$ **forall** $AC \in ACs$
15 $\quad\quad As \leftarrow AC \cup PO(\pi, \delta)$

16 **if** $\exists \alpha : \alpha \in As, \alpha \in xor(\alpha_1, \alpha_2), xor(\alpha_1, \alpha_2) \in PO(\pi, \delta)$ **then** draw task α
17 **if** $\exists \alpha : \alpha \in As, atomic(\alpha), \alpha \in PP(\pi, \delta)$ **then** draw (start event and) task α
18 **if** $\exists \alpha : \alpha \in As, \alpha \in xor(\alpha_1, \alpha_2), xor(\alpha_1, \alpha_2) \in PP(\pi, \delta)$
19 **then** draw (start event and) task α
20 **if** $\exists \alpha_1, \alpha_2 : \alpha_1 \in As, \alpha_2 \in As, and(\alpha_1, \alpha_2) \in PP(\pi, \delta)$
21 **then** draw (start event and) tasks α_1, α_2 in parallel
22 **if** $OTP(\pi, \delta) \cup PTP(\pi, \delta) \neq \emptyset$ **then** (draw event gateway)
23 **if** $\exists \alpha : \alpha \in OTP(\pi, \delta), \alpha \in As$ **then** draw start/intermediate event α
24 **if** $\exists \alpha : \alpha \in OTP(\pi, \delta), \alpha \notin As$ **then** draw intermediate timeout event α
25 **if** $\exists \alpha : \alpha \in PTP(\pi, \delta), \alpha \in As$ **then** draw start/intermediate event α
26 perform activities As
27 drawControlFlow(π, δ)
28 revoke activities As
29 **end forall**
30 **else**
31 **if** $\{\nu : \nu \in VTM(\pi, \delta)\} \neq \emptyset$ **then** draw error end event
32 **if** $\neg \exists \nu : \nu \in VTM(\pi, \delta)$ **then** (draw end event)
33 **end if**

6 Conclusion

The sequence and timing constraints on the activities in business processes are an important aspect of compliance. In this paper, we present an approach that declaratively captures these constraints, with the purpose of (re)using them in business process design. Rather than modeling precedence relations for one business partner in a particular process, PENELOPE focuses on what *can* or *must* be done at certain points in time, by *all* business partners, in order to achieve their business goals, without considering one business process model in particular.

The third party perspective on the modeling of sequence and timing constraints makes it possible for external deontic assignments to be shared among process designers of different organizations. In addition, deontic assignments are autonomous units of business logic that hold in general rather than for one particular business process. A such, changes to sequence and timing aspects in business policy and regulations can be translated into temporal deontic assignments that potentially constrain multiple business process models. The generation of individual process models from temporal deontic assignments is not intended as a means for process execution, but to be used by the process designer for verification and validation. Moreover, this explicit generation of control flow can be used to identify the *freedom of choice* that is left by the sequence and timing constraints. It is up to the designer of the process to decide whether this freedom of choice is to be filled in at design-time or at runtime. In addition, the automatic generation of control flow contains an enumeration of all possible violations of obligations by other agents that allows the process designer to anticipate exceptions in current business process design.

References

1. Goedertier, S., Vanthienen, J.: Compliant and Flexible Business Processes with Business Rules. In: CAiSE'06 Workshop BPMDS'06, Proceedings. (2006) forthcoming.
2. Reijers, H.A., Rigter, J.H.M., van der Aalst, W.M.P.: The case handling case. Int. J. Cooperative Inf. Syst. **12**(3) (2003) 365–391
3. Bons, R.W.H., Lee, R.M., Wagenaar, R.W., Wrigley, C.D.: Modelling inter-organizational trade using documentary petri nets. In: HICSS (3). (1995) 189–198
4. Bussler, C.: The role of B2B protocols in inter-enterprise process execution. In: TES '01: Proceedings of the Second International Workshop on Technologies for E-Services, London, UK, Springer-Verlag (2001) 16–29
5. Marín, R.H., Sartor, G.: Time and norms: a formalisation in the event-calculus. In: ICAIL '99: Proceedings of the 7th international conference on Artificial intelligence and law, New York, NY, USA, ACM Press (1999) 90–99
6. Yolum, P., Singh, M.P.: Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artificial Intelligence **42**(1-3) (2004) 227–253
7. Knottenbelt, J., Clark, K.: An architecture for contract-based communicating agents. In: Proceedings of the 2nd Europ. Workshop on Multi-Agent Sys. (2004)
8. Governatori, G.: Representing business contracts in *uleml*. Int. J. Cooperative Inf. Syst. **14**(2-3) (2005) 181–216
9. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. **47**(2) (2003) 237–267
10. Føllesdal, D., Hilpinen, R.: Deontic logic: An introduction. In Hilpinen, R., ed.: Deontic Logic: Introductory and Systematic Readings. D. Reidel Publishing Company, Dordrecht (1971) 1–35
11. Clarke, E.M., Grumberg, O., Long, D.E.: Verification tools for finite-state concurrent systems. In de Bakker, J.W., de Roever, W.P., Rozenberg, G., eds.: REX School/Symposium. Volume 803 of LNCS., Springer (1993) 124–175
12. Broersen, J., Dignum, F., Dignum, V., Meyer, J.J.C.: Designing a deontic logic of deadlines. In Lomuscio, A., Nute, D., eds.: DEON. Volume 3065 of LNCS., Springer (2004) 43–56
13. Shanahan, M.: Solving the frame problem: a mathematical investigation of the common sense law of inertia. MIT Press, Cambridge, MA, USA (1997)
14. Goedertier, S., Vanthienen, J.: Business Rules for Compliant Business Process Models. In: Proceeding of the 9th International Conference on Business Information Systems (BIS 2006). Volume P-85 of LNI., GI (2006)
15. Yolum, P.: Towards design tools for protocol development. In: AAMAS '05, New York, NY, USA, ACM Press (2005) 99–105